CSS
Geeky Changer CSS

# Formulaires

Lors du TD précédent, nous avons vu comment créer une base de données, l'administrer via phpMyAdmin mais également comment l'utiliser depuis un programme PHP pour insérer des données dans la base puis afficher ces données dans une page Web.

Dans ce TD nous allons voir comment réaliser un module de gestion d'utilisateurs pour un site Web dynamique en utilisant **une base de données et le mécanisme des sessions de PHP**. Pour cela, nous allons vous guider pas à pas dans la réalisation des différents éléments de ce module.

Ce TD suivra le plan suivant :

1. découverte des formulaires HTML ;
2. création d'un mécanisme de login pour votre site ;
3. création d'un mécanisme de gestion des utilisateurs pour votre site ;
4. restrictions d'accès à certaines parties de votre site en fonction des types d'utilisateurs.

## 1. À la découverte des formulaires HTML

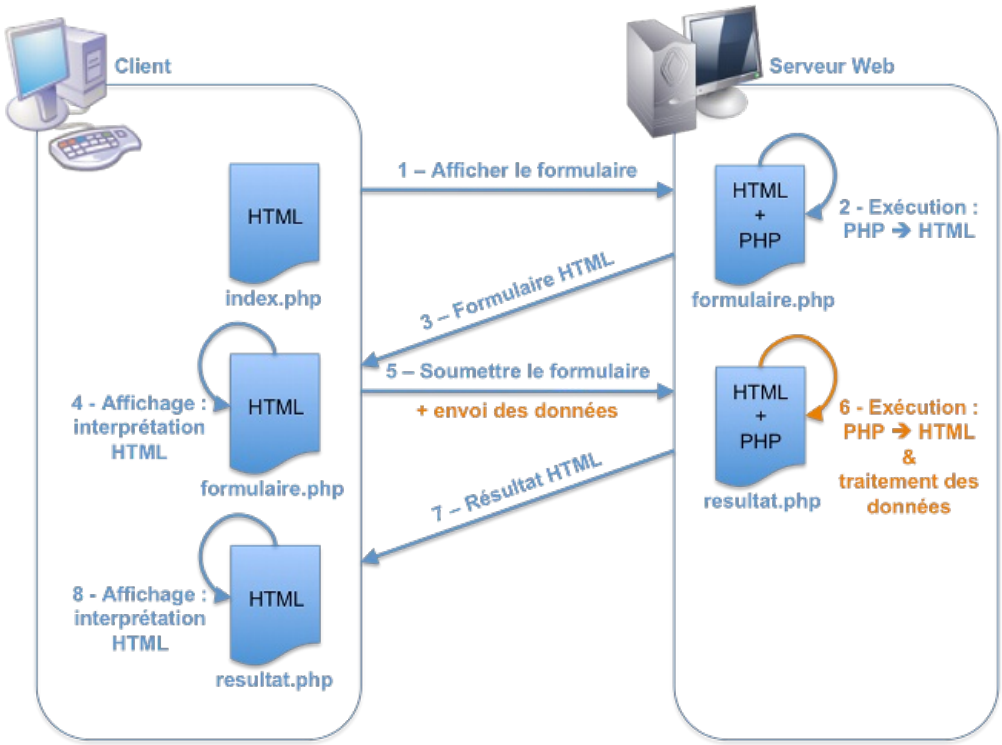
Les formulaires permettent à l'utilisateur de saisir des données et de les envoyer au serveur, qui utilise ces données pour générer le contenu de la page Web de façon personnalisée. Ainsi par exemple, un formulaire peut permettre à l'utilisateur de saisir son âge, et cette donnée est utilisée par le serveur qui pourra masquer certaines parties d'une page Web qui ne sont pas accessible aux mineurs. Dans cette partie du TD, nous allons voir comment créer des formulaires HTML et envoyer les entrées de l'utilisateur au serveur.

### 1.1 Le plus simple des formulaires

Le fonctionnement des formulaires est le suivant :

- Une première page présente à l'utilisateur le formulaire à proprement parler, c'est-à-dire un certain nombre de champs à remplir (texte libre, case à cocher, etc.). Cette page indique en outre quelle page doit être chargée lors de la validation du formulaire.
- Cette seconde page reçoit le contenu des champs du formulaire et affiche un résultat. Ce résultat peut être un simple message (statique) de confirmation, ou bien peut être généré dynamiquement (entre autres par un script PHP), à partir des données du formulaire fournies.

Le schéma suivant illustre ce principe :




Dans un premier temps, nous allons créer deux pages en HTML pur ; nous ajouterons du PHP ensuite. Le formulaire est donné ci-dessous.

solutions/formulaire1.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Formulaire</title>
  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
</head>

<body>
  <h1>Mon premier formulaire !</h1>

  <form action="resultat1.php" method="get">
    <p>Prénom : <input name="prenom" /></p>
    <p>Genre :
      <input checked="true" name="genre" value="f" type="radio" />F
      <input name="genre" value="m" type="radio" />M
    </p>
    <p><input type="submit" value="Valider" /></p>
  </form>
</body>

</html>
```

Le formulaire lui-même est entouré de balises `<form> ... </form>`. Dans la balise ouvrante, l'attribut `action` indique le nom de la page à afficher lors de l'envoi des données. Nous parlerons de l'attribut `method` plus tard. À l'intérieur du formulaire lui-même se trouvent des textes et balises divers, mais surtout les *champs du formulaire*, ici deux balises `input` : la première constitue un champ de texte brut, la seconde un choix parmi des alternatives (composants dits *boutons radio*).

Les champs peuvent être des zones d'entrée de texte à une (`<input type="text" ... />`) ou plusieurs lignes (`<textarea ... />`), de menus déroulants (`<select ... > <option> ... </option> ... </select>`) à choix unique ou multiple (`multiple="multiple"`), de cases à cocher (*check boxes*, `<input type="checkbox" ... />`), de boutons radio (*radio buttons*, `<input type="radio" ... />`), de champs de choix de fichier à télécharger sur le serveur, et *last, but not least*, de boutons déclenchant la remise à zéro des entrées effectuées (`<input type="reset" ... />`) ou l'envoi des données au serveur (`<input type="submit" ... />`). Pour plus de détails, voir la section de SELFHTML dédiée aux formulaires.

Pour le moment, la page de résultat affichera juste un message de confirmation :

solutions/resultat1.php

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>
  <title>Formulaire</title>
  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8" />
</head>

<body>
  <p>Formulaire soumis.</p>
</body>

</html>
```

➔ Essayez le formulaire ! Comment les valeurs des champs sont-elles transmises à la page de résultat ?

**Astuce :** Sur la page de résultat, vous pouvez remarquer que l'URL se termine par une indication du type `?prenom=Tim&genre=m`.

## 1.2 Traitement sur les données

Nous allons maintenant voir comment accéder à ces champs depuis du code PHP inséré dans la page de résultat. Pour cela, il suffit de consulter le tableau `$_GET` (voir le TD n°2) :

```
<?php
  echo "Bonjour, " . ($_GET["genre"] == "f" ? "Mlle " : "M. ") . $_GET["prenom"] . " !";
?>
```

➔ Essayez le formulaire ! Vous pouvez vous amuser à changer à la main la valeur des champs transmis dans l'URL, et observer l'effet sur la page de résultat.

Dans cette approche, le contenu des champs du formulaire apparaît dans l'URL de la page résultat. Ce n'est pas toujours souhaitable, par exemple lorsqu'on saisit un mot de passe dans le formulaire. Il existe une variante, dans laquelle les données sont transmises lors de la requête vers la page résultat, mais pas dans l'URL. Il suffit de changer l'attribut `method` du formulaire en `post` (à la place de `get`), et d'aller chercher les données dans le tableau `$_POST` à la place de `$_GET` dans la page résultat.

➔ Installer l'exemple fourni sur votre ordinateur (code formulaire, code résultat), et modifiez ce code pour qu'il fonctionne en mode POST, et non plus en mode GET.

## 1.3 Une page unique pour le formulaire et le résultat

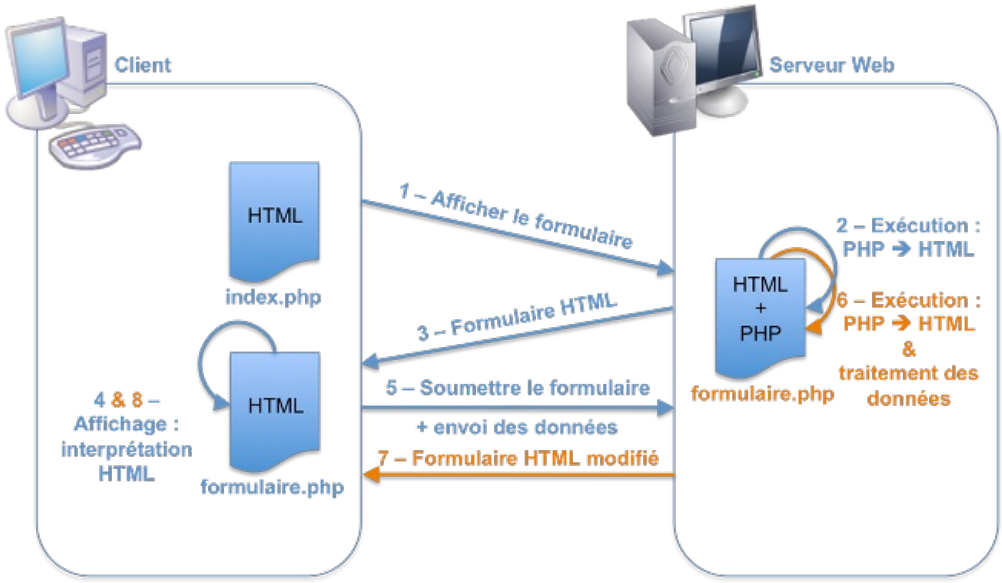
Dans l'exemple précédent, on ne traite pas le cas où l'un des deux champs n'est pas rempli. Dans ce cas, il serait souhaitable de réafficher le formulaire, en conservant le contenu des champs qui avaient été correctement remplis. Comme il serait peu intelligent de dupliquer le formulaire dans deux fichiers différents, nous allons utiliser un unique fichier, pour le formulaire et pour le résultat. Au début de la page, nous choisirons d'afficher soit le formulaire, soit la page de résultat, avec une condition du type :

```
if(isset($_GET["prenom"]) && $_GET["prenom"] != "" && isset($_GET["genre"])) {
  // code de la page résultat
} else {
  // code du formulaire
}
```

On affiche la page résultat si tous les champs obligatoires du formulaire ont été remplis, sinon on réaffiche le formulaire, dans lequel on pré-remplit les champs dont la valeur a déjà été saisie correctement. De cette façon, l'utilisateur ne perd pas sa saisie, même si elle est partielle. On peut procéder de la façon suivante :

```
<input name="prenom" value="<?php if(isset($_GET["prenom"])) echo $_GET["prenom"]; ?>" />
```

Le schéma suivant illustre le principe expliqué ci-dessus, dans lequel un seul fichier PHP réalise à la fois l'affichage du formulaire et son traitement :




➔ À vous de jouer ! Modifiez le code de l'exercice précédent de manière à ce que le formulaire soit affiché pré-rempli dans le cas où un champ est manquant.

Voici une solution à l'œuvre : formulaire3.php.

# 2. Un formulaire de connexion (et déconnexion)

Dans le TD n°3, vous avez créé une table de base de données contenant des comptes utilisateurs, notamment le login et le mot de passe de chaque utilisateur. Nous allons maintenant utiliser cette information pour mettre en place un mécanisme de connexion (*login*), de façon à restreindre l'accès au site. Les utilisateurs non connectés pourront n'avoir accès qu'à certaines pages du site. Par exemple, ils pourraient être autorisés à consulter les messages d'un forum, mais pas à envoyer de nouveau message.

Toutes les pages du site contiendront un formulaire de connexion ou bien de déconnexion, selon que l'utilisateur est, respectivement, non connecté ou connecté. Commençons par le formulaire de connexion.

## 2.1 Génération du formulaire de connexion

Le formulaire de login figurera dans toutes les pages. De façon à clarifier le code, nous allons écrire une fonction de génération du formulaire de login, que nous rangerons dans un fichier `printForms.php`. Plus tard, nous ajouterons à ce fichier la fonction de génération du formulaire de déconnexion.

Un formulaire de login doit comporter :

- un champ de saisie du nom d'utilisateur (login) ;
- un champ de saisie du mot de passe (l'affichage des caractères doit être masqué en utilisant un champ de type `password`) ;
- un bouton pour valider la saisie.

C'est la page centrale de notre site, `index.php`, qui devra appeler `printLoginForm` pour l'afficher sur toutes les pages avant connexion. En retour, nous allons convenir que le formulaire de login appellera `index.php`, mais avec un paramètre GET `action=login`, de façon à lui signifier qu'il faut traiter une demande de connexion.

➔ Comment faire en sorte que vous puissiez à la fois appeler une page avec un paramètre GET, et envoyer les résultats d'un formulaire ? Cela permet-il de s'assurer que le mot de passe saisi dans le formulaire de login n'apparaît jamais en clair à l'écran ?

➔ Dans `printForms.php`, écrivez une fonction `printLoginForm` (sans paramètre) qui affiche un formulaire de connexion.

## 2.2 Traitement du formulaire de connexion

Lorsque l'utilisateur validera le formulaire de connexion, la page principale `index.php` sera donc appelée, avec le

paramètre GET `action=login`, et les champs du formulaire seront transmis. La page principale doit alors :

1. vérifier si l'utilisateur est bien présent dans la table `utilisateurs` ;
2. vérifier qu'il possède bien le mot de passe indiqué ;
3. effectuer la connexion proprement dite.

Pour les points 1 et 2, cela va se traduire par des requêtes SQL à effectuer sur la base de données, comme vous l'avez appris dans le TD3. Pour le point 3, nous avons envie que, pendant toute la suite de la navigation sur le site, à chaque demande d'une nouvelle page, PHP puisse se souvenir de certaines informations, en l'occurence que l'utilisateur est bien connecté et son login. Nous aurions donc envie de créer une *variable persistante*. Pour ce faire, PHP nous propose le mécanisme des *sessions*. Avant de continuer sur le formulaire de connexion, voyons de quoi il s'agit...

### 2.2.1. Variables de session

Les variables de session sont des variables qui se transmettent de page en page pour chaque utilisateur au cours de sa visite, et cela de façon invisible pour le client — en fait, seul le serveur est à même de manipuler les variables de session, ce qui en fait un mécanisme parfait pour mémoriser, par exemple, si la vérification d'un couple utilisateur/mot de passe a été correctement effectuée.

Lorsqu'un utilisateur va se connecter à un site utilisant des sessions, un identifiant, l'*ID de session*, unique, généré aléatoirement, va lui être attribué. C'est cette identification qui va suivre l'utilisateur tout au long de sa visite sur le site et qui permettra à PHP de distinguer un client *authentifié* de celui qui ne l'est pas. Les « cookies » ou témoins sont en général utilisés pour permettre de stocker cet identifiant.

En utilisant XAMPP comme vous le faites, vous ne devez rien configurer pour pouvoir utiliser cette fonctionnalité. Notez cependant que ce n'est pas le cas pour tous les hébergeurs (par exemple, si vous avez un site chez Free, vous devez créer un sous-répertoire `sessions` à la racine du répertoire contenant votre site. C'est bon à savoir pour l'après-modex !)

Pour ouvrir une session, il vous faut appeler la fonction `session_start()`. Cette fonction initialise la session de l'internaute.

**Notez qu'AUCUNE SORTIE, ni même des espaces ou des lignes blanches, n'est tolérée AVANT l'usage de cette fonction `session_start()`. Pour plus d'informations, regardez là.** Nous vous recommandons donc de faire figurer l'appel à `session_start()` tout au début de `index.php`. PHP effectue alors les opérations suivantes :

- PHP essaie de récupérer un ID de session pour l'utilisateur.
- Si PHP trouve un ID de session, il restitue la session et les données associées.
- Si PHP ne trouve pas d'ID de session (généralement parce que l'utilisateur vient d'arriver sur le site), il génère un ID de session et une nouvelle session est créée.

Pour stocker les données (dans notre cas précis, un indicateur précisant si l'identification a été correcte, que l'on peut compléter par le nom complet de l'utilisateur ainsi que son login), on utilise la variable (tableau) `$_SESSION`. Ainsi pour définir et affecter une variable de session, on utilisera une instruction du type :

```
$_SESSION['mavariable'] = valeur;
```

La valeur de la variable peut être obtenue par :

```
$_SESSION['mavariable']
```

Une dernière commande utile est `session_destroy()` qui permet de fermer une session. C'est typiquement la commande à appeler lors de la déconnexion. Avant de l'appeler, il convient cependant d'effacer toutes les variables de session. On emploiera donc le code suivant :

```
session_unset();
session_destroy();
```

Une méthode classique d'attaque d'un site web consiste à accéder à la session d'un autre utilisateur. Ces problèmes seront évoqués prochainement en amphi mais nous allons dès maintenant nous prémunir d'une telle attaque, ou du moins la rendre plus difficile.

Pour cela, nous allons associer à notre session un nom quelconque (changez-le ci-dessous) avant de la créer. De plus, nous la regénérerons afin de rendre une attaque un petit peu plus dure.

### 2.2.2. Création de la session

➔ Ajoutez comme *première* ligne de `index.php` le code suivant (en modifiant sa première ligne) :

```
<?php
    session_name("NomDeSessionAModifierSelonVotreGout" );
    // ne pas mettre d'espace dans le nom de session !
    session_start();
    if (!isset($_SESSION['initiated'])) {
        session_regenerate_id();
        $_SESSION['initiated'] = true;
    }
    // Décommenter la ligne suivante pour afficher le tableau $_SESSION pour le debuggage
    // print_r($_SESSION);
?>
```

Pour des raisons de debuggage, on pourra décommenter la dernière ligne, affichant ainsi le contenu du tableau associatif `$_SESSION`.

Lors de la déconnexion de l'utilisateur, nous devrons *détruire* la session. Nous nous en occuperons plus tard.

### 2.2.3. Stockage de données dans la session

Nous vous proposons de stocker dans une variable de session, que nous appellerons `loggedIn`, le fait que l'utilisateur soit logué : `$_SESSION['loggedIn']` sera défini si et seulement si l'utilisateur est connecté. Nous allons écrire dans un script `logInOut.php` le code servant à la connexion et à la déconnexion des utilisateurs.

➔ Dans `logInOut.php`, écrivez une fonction `logIn` (sans paramètre), qui vérifie si l'utilisateur (passé en paramètre du formulaire de login) est bien présent dans la base (point 1), si le mot de passe fourni est le bon (point 2) et effectue le login (point 3). En cas d'échec, `logIn` ne doit bien sûr pas effectuer le point 3...

➔ Pendant que vous y êtes, ajoutez au même fichier une fonction `logOut` (sans paramètre) qui permettra d'effectuer la déconnexion d'un utilisateur. Nous utiliserons `logOut` un peu plus tard.

### 2.2.4. Modification de l'affichage en fonction des données de la session

À ce stade, vous allez pouvoir mettre en place l'affichage et la gestion du formulaire de login dans `index.php`, à l'aide d'un code du genre suivant :

```
<?php /* ATTENTION, le < doit être le premier caractère du fichier */
... ouverture de session ...

if($_GET["action"] == "login") {
 logIn();
}

// code de sélection des pages, comme précédemment
if(isset($_GET[page])) {
 $askedPage = ...
 ...
}

if($_SESSION["loggedIn"]) {
 // tout à l'heure on affichera le formulaire de déconnexion
} else {
 printLoginForm();
}

... code d'affichage des pages, comme précédemment ...
?>
```

➔ Implémentez la gestion du login.

### 2.2.5. Amélioration

Avec le système actuel, lorsqu'on valide le formulaire de login, la variable `$_GET["page"]` n'est pas correctement définie, et donc on se retrouve sur la page par défaut. Ce que l'on aimerait, c'est certes effectuer le login, mais se retrouver ensuite sur la page à partir de laquelle on a validé le formulaire.

Pour y parvenir, on peut modifier la fonction `printLoginForm` dans `printForms.php`, de façon qu'elle transmette le bon paramètre GET. Pour cela, elle aura elle-même besoin de recevoir ce paramètre ; elle se contentera de l'ajouter à l'attribut `action`. Le script appelant saura lui transmettre la bonne valeur de ce paramètre, il la tirera de sa propre variable de nom de page (la variable `$askedPage` et non la chaîne stockée dans `$_GET` pour tenir compte de la page par défaut et éviter les noms de page de pirate). Le code ci-dessus devient donc :

```
...
if($_SESSION["loggedIn"]) {
 // tout à l'heure on affichera le formulaire de déconnexion
} else {
 printLoginForm($askedPage);
}
...
```

### 2.3 Déconnexion : à vous de jouer

Avec ce que vous avez réalisé jusqu'ici, vos utilisateurs peuvent se connecter, mais pas se déconnecter. Voici ce qu'il vous reste à faire pour mettre en place la déconnexion :

- ajouter une fonction `printLogoutForm` dans `printForms.php`, sur le modèle de `printLoginForm` ;
- dans `index.php`, commander l'affichage de ce formulaire de déconnexion lorsque l'utilisateur est logué ;
- lorsque `index.php` recevra une demande de déconnexion, elle devra appeler la fonction `logOut` que nous avons écrite un peu plus haut dans `logInOut.php`.

➔ Implémentez le mécanisme de déconnexion comme suggéré ci-dessus.

## 3. Gestion des utilisateurs

Jusqu'à présent, vous avez supposé que votre base de données contenait quelques utilisateurs, rentrés à la main la dernière fois à l'aide de phpMyAdmin. Maintenant, nous allons voir comment ajouter à votre site des pages pour :

- permettre à un nouvel utilisateur de s'enregistrer ;
- permettre à un utilisateur de modifier son mot de passe ;
- permettre à un utilisateur de supprimer son compte.

### 3.1 Formulaire d'enregistrement d'un utilisateur : « Je veux ma carte ! »

Nous allons d'abord créer une page, que nous appellerons `register.php`, pour permettre à un utilisateur de s'enregistrer et de créer son compte. Elle doit contenir le code HTML d'un formulaire avec les éléments suivants :

- des champs de type `texte` pour le login, le nom complet et l'adresse e-mail ;
- des champs de type `password` pour le mot de passe et sa confirmation ;
- une liste déroulante pour permettre à l'utilisateur de choisir parmi les feuilles de style disponibles (vous pourrez par exemple réutiliser les feuilles de style des TD) ;
- et, bien sûr, un bouton « valider ».

Nous utiliserons la technique vue en première partie : une seule page pour le formulaire et le traitement de ses résultats. Aussi, dans le formulaire, vous spécifierez `action="<?php echo $_SERVER["REQUEST_URI"]; ?>"` de façon à ce que la même URL soit utilisée lorsque l'utilisateur valide le formulaire.

➔ Tout d'abord, créez cette nouvelle page, qui pour le moment contiendra juste le formulaire.

➔ Après avoir créé la page, vérifiez qu'elle s'affiche correctement dans le navigateur.

Maintenant, nous voulons effectuer un traitement lorsque l'utilisateur valide le formulaire : créer un nouveau compte ! En appliquant la technique de la première partie, vous allez donc restructurer la page de la façon suivante :

```
$form_values_valid=FALSE;

if(isset($_GET["login"]) && $_GET["login"] != "" &&
   isset($_GET["email"]) && $_GET["email"] != "" &&
   ...) {  // tous les champs requis cités ici
  // code de traitement, à écrire maintenant
```

```
  // si le traitement réussi, on passe $form_value_valid à TRUE
}

if (!$form_values_valid) {
  // code du formulaire, qui vient d'être écrit
  // où les valeurs par défaut des champs sont pris dans le tableau $_GET
  // ($_GET["champ"] est vide si non défini)
}
```

➔ Modifiez la structure pour correspondre à ce modèle. Pour le moment, vous pouvez vous contenter d'afficher un simple message dans la partie « traitement ».

Le traitement du formulaire d'enregistrement d'un utilisateur doit effectuer les actions suivantes :

1. vérifier si le login n'existe pas encore dans la base ;
2. vérifier si les deux mots de passe entrés sont identiques ;
3. ajouter le compte utilisateur dans la base de données ;

Avant de nous lancer tête baissée dans le codage, prenons le temps de réfléchir un peu... Vous avez déjà écrit une fonction `logIn` qui :

1. vérifie si un utilisateur est présent dans la base ;
2. vérifie si son mot de passe correspond à un mot de passe donné ;
3. et enfin effectue le login.

La première étape du traitment du login et du formulaire d'enregistrement sont donc identiques, en conséquence nous pouvons factoriser cette partie...

➔ Écrire une fonction `isUser($user)` (à mettre dans `auth.php`) qui renvoie `TRUE` si l'utilisateur `$user` existe, `FALSE` sinon.

➔ Modifier la fonction `logIn` (dans `logInOut.php`) de façon à ce qu'elle utilise `isUser`. Testez soigneusement le formulaire de login ainsi modifié de façon à vous convaincre du bon fonctionnement de `isUser`.

➔ Écrire le traitement du formulaire d'enregistrement, en utilisant `isUser`.

➔ En cas d'échec de l'enregistrement, l'utilisateur souhaite ne pas devoir remplir à nouveau tous les champs du formulaire. Faites en sorte qu'en cas de réaffichage du formulaire, le login soit repris, en utilisant la technique expliquée ci-dessus. (Il n'est pas d'usage de reprendre les mots de passe, qui de toute façon ne seraient pas affichés.)

### 3.2 Formulaire de changement de mot de passe : « Au secours ! Ma mère a découvert mon mot de passe ! »

Eh oui, ça arrive — et imaginons que ce ce mot de passe soit celui donnant accès aux photos de vous prises au dernier Point Gamma, clairement vous avez envie de changer rapidement de mot de passe...

Nous allons donc concevoir une nouvelle page `changepassword`, qui contiendra un formulaire muni :

- optionnellement (selon vos goûts) d'un champ de type `texte` pour le login, sinon il sera pris dans la variable de session ;
- de champs de type `password` pour l'ancien mot de passe, le nouveau mot de passe et sa confirmation ;
- et, bien sûr, d'un bouton « valider ».

➔ Créez ce formulaire et la page correspondante.

Le traitement correspondant doit comprendre, dans cet ordre, les actions suivantes :

1. vérifier que les deux entrées pour le nouveau mot de passe sont identiques ;
2. vérifier qu'il existe dans la base un utilisateur dont le login est celui entré dans le formulaire ou dans la variable de session ;
3. vérifier que l'ancien mot de passe entré dans le formulaire est identique au mot de passe associé dans la base à l'utilisateur en question ;
4. mettre à jour le mot de passe haché (fonction `SHA1($passwd)`) dans la base de données.

Dans chacun des points 1 à 3, il faut arrêter sans modifier la base en cas d'erreur, et réafficher le formulaire.

Notez que normalement, dans le TD précédent, vous avez déclaré le champ `login` en tant que clé primaire, de façon à ce que plusieurs utilisateurs ne puissent pas avoir le même login ! Vous ne devriez donc pas modifier le mot de passe d'un autre utilisateur...

Le point 1 ne présente pas de difficulté. Vous venez d'écrire une fonction `isUser` pour traiter le point 2.

Quant au point 3... Vous avez déjà traité ce problème dans le formulaire de login ! Comme précédemment, nous vous proposons donc de factoriser ce traitement.

➔ Ajoutez à `auth.php` une fonction `isPasswordCorrect($user, $password)` qui renvoie `TRUE` si `$password` est le mot de passe de l'utilisateur `$user` (dont on suppose qu'il existe), `FALSE` sinon. Modifiez `logIn` pour utiliser `isPasswordCorrect`.

Le point 4 constitue la partie vraiment nouvelle : comment mettre à jour les informations existantes dans la base ? Pour cela, nous allons créer une expression SQL du type :

```
UPDATE `nom_table` SET `champ1`='new_val1', `champ2`='new_val2' WHERE `champ3`='old_val3';
```

Vous pouvez utiliser l'interface phpMyAdmin pour voir quelles requêtes elle génère pour changer des entrées de la table.

À la lecture de cette expression, MySQL va remplacer, dans tous les enregistrements de la table `nom_table` qui ont leur `champ3` égal à `old_val3`, la valeur du `champ1` par `new_val1` et la valeur du `champ2` par `new_val2`. Les champs à modifier, à gauche du `WHERE`, sont séparés par des virgules ; la condition à droite du `WHERE` permet de sélectionner les enregistrements sur lesquels on veut faire les modifications.

Bien sûr, vous devez ici adapter l'expression au cas de votre formulaire...

➔ Écrivez le traitement associé au formulaire de changement de mot de passe.

➔ Vérifiez son bon fonctionnement, et notamment que :

- vous ne pouvez changer le mot de passe que pour les utilisateurs dont vous connaissez le login et le mot de passe, et dont les deux entrées pour le nouveau mot de passe sont identiques,
- les mots de passe sont effectivement changés dans la base de données et que les anciennes données qui ne doivent pas être modifiées (nom complet, email, etc.), sont bien conservées.

## 3.3 Formulaire de suppression d'un utilisateur : « À l'aide ! Je veux partir, c'est tellement mieux ailleurs... »

Même les meilleures choses ont une fin, arrivera tôt ou tard le jour où l'un de vos utilisateurs voudra se désinscrire de votre site.

Nous allons créer une page `deleteuser`, qui contiendra un formulaire comprenant :

- optionnellement (toujours selon vos goûts) d'un champ de type `texte` pour le login, sinon il sera pris dans la variable de session ;
- un champ de type `password` pour le mot de passe ;
- et, bien sûr, un bouton « valider ».

➔ Créez cette nouvelle page et le formulaire.

Le traitement associé doit effectuer, et dans cet ordre, les actions suivantes :

1. vérifier que l'utilisateur dont le login est donné existe ;
2. vérifier que le mot de passe entré dans le formulaire est bien identique à celui stocké dans la base de données ;
3. supprimer l'enregistrement correspondant à cet utilisateur.

Dans chacun des points 1 à 2, arrêtez sans modifier la base lorsque la condition pour continuer n'est pas remplie, et réaffichez le formulaire.

Les fonctions `isUser` et `isPasswordCorrect` doivent vous permettre de traiter les points 1 et 2. Seul le point 3 est nouveau : comment supprimer un enregistrement de la base ? Pour cela, nous allons créer une expression SQL de la forme :

```
DELETE FROM `nom_table` WHERE `champx`='valx';
```

Cette requête permet de supprimer de la table `nom_table` toutes les entrées telles que `champx` est égal à `valx`.

➔ Écrivez le code de suppression d'un utilisateur.

➔ Vérifiez que le traitement s'effectue correctement, en particulier :

- que vous ne supprimez que les utilisateurs dont vous avez saisi le login et le mot de passe ;
- que les utilisateurs supprimés le sont effectivement, c'est-à-dire qu'ils ne peuvent plus se connecter et que l'entrée correspondante a disparu de la table (facilement vérifiable avec phpMyAdmin).

### 3.4 Mettre en place des liens vers nos formulaires

➔ Modifiez `index.php` de sorte qu'un utilisateur non connecté voie un lien vers le formulaire de connexion.

➔ Modifiez `index.php` de sorte qu'un utilisateur connecté voie un lien vers le formulaire de changement de mot de passe et de suppression de son compte.

## 4. Utilisation de l'identification des utilisateurs

### 4.1 « Top secret » : réserver certaines pages seulement aux utilisateurs enregistrés

Maintenant que nous savons gérer l'identification d'un utilisateur, servons-nous en ! Pour cela, en haut des pages `content_*` réservées aux utilisateurs enregistrés, nous allons ajouter le code suivant :

```
if (!isset($_SESSION['loggedIn'])) {
  echo "Page non autorisée";
  return;
}
```

L'appel `return` arrête l'exécution du code de la page et retourne à `index.php` pour afficher proprement la fin de la page principal.

### 4.2 What else ?

➔ Vous pouvez également faire en sorte qu'un utilisateur non connecté voie le site avec une feuille de style par défaut, mais qu'un utilisateur connecté le voie avec la feuille de style qu'il a choisie.

➔ De même, certaines pages pourraient afficher des contenus différents selon que l'utilisateur est connecté ou non.

Vous disposez maintenant de toutes les briques de base pour vous lancer dans la création de contenu de votre propre site... À vous de jouer !